2

US ARMY
LABORATORY COMMAND

AD-A214 811

CONTRACT REPORT NUMBER 8-89
PREPARED FOR THE

HUMAN ENGINEERING LABORATORY

BY

COMMUNICATIONS DESIGN CENTER
CARNEGIE MELLON UNIVERSITY
PITTSBURGH, PA 15213

EXECUTIVE SUMMARY
FOR
DESIGNING ONLINE HELP SYSTEMS:
REPORT OF YEAR 1 ACTIVITY

4 April 1987

DAAA1-86-K-0019AC85

DTIC
ELECTE
NOV 27 1989
S B D

Approved for public release;
distribution is unlimited.

ABERDEEN PROVING GROUND, MARYLAND 21005-5001

Executive Summary
for
Designing Online Help Systems: Report of Year 1 Activity

prepared for

United States Army Laboratory Command
Human Engineering Laboratory
Aberdeen Proving Grounds, MD 21005

prepared by

Thomas M. Duffy, James Palmer, Kathleen Gomoll,
Thomas Gomoll, Jessica Richards-Palmquist, and John Trumble

4 April 1987

Communications Design Center
Carnegie Mellon University

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS None.
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Carnegie-Mellon University 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION U.S. Army Human Engineering Laboratory
6c. ADDRESS (City, State, and ZIP Code) Communications Design Center Carnegie-Mellon University Pittsburgh, PA 15213	7b. ADDRESS (City, State, and ZIP Code) ATTN: SLCHE-CS Aberdeen Proving Ground, MD 21005-5001
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER Contract # DAAA1-86-K-0019AC85
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

~~Help Design Software Project (Unclassified)~~ See Cover

12. PERSONAL AUTHOR(S) Thomas M Duffy, James Palmer, Brad Mehlenbacher, Guojun Zhang, Ann Aaron, Maria Truschel, Karen Denchfield

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM 4 Apr 87 TO 30 Nov 88	1988, Nov. 30	270

16. SUPPLEMENTARY NOTATION

This task was performed under the contract title: On-Line Help for Interactive Systems

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	On-Line Help, HELP Systems, human-computer interface, human factors

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Interactive computer systems were once used only by highly-trained specialists. Today, complex computer systems are used by people with varying levels of skill and experience. Adequate on-line help systems must be designed so new or infrequent users will be able to effectively use the systems without lengthy training or experimentation. The purpose of this project was to perform a systematic investigation of on-line help and to develop guidelines for the design of on-line help systems for interactive computer systems. Three topics were hypothesized and investigated:

1. The help database should be arranged so that it can be efficiently maintained without sacrificing the capacity for many kinds of access and it should appear consistently so that entries can be added or amended easily. Alternate organizational strategies that balanced user-friendliness with ease of maintenance were developed and tested.

(continued on reverse side)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. (Continued from first page)

2. Each entry in the database should be structured for effective scanning and interpretation. Alternate presentation formats for on-line help information were evaluated. Optimal display strategies are reported.

3. The type of information in the database should be specific to the users for which the on-line help system is designed. Procedures for assessing the kinds of information that will be useful were developed. Access channels to the on-line help database were designed.

Accession For		
NTIS GRA&I	☑	
DTIC TAB	☐	
Unannounced	☐	
Justification		
stribution/		
Availability Codes		
Dist	Avail and/or Special	
A-1		

Our contract, Online Documentation for Interactive Systems (DAAA15-86-K-0019) began March 7, 1986. This report summarizes our progress during the past year.

The general goals of this contract, as stated in the original proposal, were to develop and evaluate alternative strategies for designing on-line help. Based on our work thus far, we have refined and expanded these goals to identify the three main project deliverables and the tasks that must be achieved to complete them:

1. A handbook of guidelines and procedures for the design of on-line help systems. While the original contract called for the development of guidelines, they were to be a by-product of an analytical literature review and an evaluation of various online help design strategies.

Our revised goal is to develop a handbook of guidelines that software engineers can use on the job. Too often, handbooks with guidelines give only a description of what the final product should look like without giving specific examples or directions on how to achieve that look. Our handbook will offer ready access to relevant and useful information.

The tasks that contribute to development of the guidelines are as follows:

1.1 A review of literature on the design of online help systems. We have compiled a list of articles related to online help and are now entering abstracts in a database program (NoteBook) that runs on either an IBM PC or Apple Macintosh. This makes the information included in the database readily accessible with most microcomputers.

The literature review process will continue for the duration of the project; as new articles pertaining to online help are published, we will add them to the database. However, the major portion of the database and a review paper discussing our findings will be completed and submitted for review by 1 November 1987.

1.2 Interviews with writers of online help systems to capture expert knowledge on the design of online help systems. The interviews reveal how technical writers differentiate between writing and designing online vs. hard copy information, and how they use existing guidelines for designing online help.

We have interviewed ten writers. A report summarizing the interview data and guidelines derived from the data will be delivered by 1 June 1987. We plan to continue these interviews and expand them to include questions about the process by which the help system is developed. This would help generate guidelines applicable to software engineers.

1.3 The establishment of help system benchmarks. To our knowledge, there is no methodology for comparing and evaluating different online help systems. Our goals are to develop a method for describing features of online help systems and a strategy for testing and comparing the effectiveness of those features. One result of this investigation will be the formulation of guidelines for applying the best features of current online help systems.

We have designed a form for recording various features of online help systems and are now reviewing the help systems for 30 different software packages. We will submit a preliminary report of our work on this task by 1 July 1987. Our work on this task will continue throughout the next year.

2. A kit of help system design tools that can be used to evaluate any online help system. The original proposal called for us to build a prototype help system that we would use to evaluate design variables. We have completed the components of that system that we need to conduct our evaluations of online help variables. However, this first year experience has led us to recognize the need for more general tools to aid in the design and test of help systems. Therefore we recommend that the tasks under this area include the review of commercial software now available for creating help systems. If these systems seem too limited we would further recommend the expansion of our specific system into a more general set of system design tools. These tools will provide an environment in which other programmers can more easily implement and test help systems.

The tasks that contribute to development of the help system design tools are as follows:

- choosing a development environment
- choosing an application program to design help for
- implementing several help systems for testing

We have already completed the first two tasks. We are using three Sun 3/50 Advanced Workstations from Sun Microsystems for all our development efforts. The help systems themselves are implemented in Franz Extended Common Lisp. We have also chosen EMACS as our application program. The third task is progressing in accordance with other task areas. Currently, we have implemented two versions of command help as well as data collection programs that record time-stamped keystroke data for both EMACS and the help system itself.

3. A series of technical reports evaluating various design strategies for online help menus, screens, and text. The main goal of this work is to construct a model of the user of online help

systems; to design an intelligent help system, we must understand how the user interacts with help systems. To gain this understanding, we will be analyzing the information processing and problem solving behaviors of different users (novice and expert) given two different menu designs and four different types of help text. The data we collect will provide input for the design of our prototype help system as well as generate guidelines for the design of help systems in general.

The tasks that contribute to the evaluation of design strategies are as follows:

3.1 Problem representation, learning and menu design. The goal of this task area is to collect data on how users represent problems and select strategies for solving those problems. Perhaps most importantly, we want to study how menu design affects the problem solving process and how that process changes with learning. To do so, we will be comparing two menus used by novices and intermediates to complete tasks at three levels of complexity. Data will be collected by 15 July 1987 and a draft report of our findings will be submitted by 15 October 1987.

3.2 Procedures for defining a menu structure. In this task area we intend to examine alternative methods for creating a help system menu structure and evaluate the effectiveness of those methods. This work will contribute to the development of procedural guidelines for structuring help menus. We will be comparing four different methodologies for creating help menus: card sort, task representation, computer scientist judgment, and document designer judgment.

We have already card sort data and are now starting to collect task representation data as described in task area 3.1. These two groups of data will be analyzed and reported by 1 October 1987. Once we have completed the menu designs for all four methods, we will evaluate the effectiveness of the different designs. We plan to begin that evaluation by 1 October 1987.

3.3 Defining the content and design of help information. This last task area focuses on the type of help information or text that is presented to the user. We have defined three types of help databases: command, task oriented, discursive. The databases will be completed by 1 June 1987 and the usability of the help texts can then be compared. The comparisons will include empirical measures of performance, such as speed, accuracy, and rate of learning. We will also collect protocols of people working with the different help texts to develop a more thorough understanding of the affects of each design on the users.

Designing Online Help Systems: Report of Year 1 Activity

prepared for

United States Army Laboratory Command
Human Engineering Laboratory
Aberdeen Proving Grounds, MD 21005

prepared by

Thomas M. Duffy, James Palmer, Kathleen Gomoll,
Thomas Gomoll, Jessica Richards-Palmquist, and John Trumble

4 April 1987

Communications Design Center
Carnegie Mellon University

Table of Contents

1 Develop a Handbook of Online Help Guidelines 2

1.1 Literature Review 4

1.2 Writing Online Information: Expert Strategies 6

1.3 Help System Benchmarks 9

2 Develop a Toolkit for Designing Help Systems 11

3 Evaluate Design Strategies 15

3.1 Problem Representation, Learning, and Menu Design 16

3.2 Procedures for Defining a Menu Structure 18

3.3 Defining the Content and Design of Help Information 20

Introduction

Our contract, Or...ie Documentation for Interactive Systems (DAAA15-86-K-0019) began March 7, 1986. This report summarizes our progress during the past year.

The general goals of this contract, as stated in the original proposal, were to develop and evaluate alternative strategies for designing online help. Based on our work thus far, we have refined and expanded these goals to identify the three main project deliverables and the tasks that must be achieved to complete them:

1. **Develop a handbook of guidelines and procedures for the design f online help systems.** While the original contract also called for guidelines, they were to be a by-product of the analytical literature review and the experimental comparisons. We now see the need to develop a handbook of guidelines for the design of online help; thus, we recommend that the handbook become our central objective.

2. **A kit of help system design tools that can be used to evaluate any online help system.** The original proposal called for us to build a prototype help system that we would use to evaluate design variables. We have completed the components of that system that we need to conduct our evaluations of online help variables. However, this first year experience has led us to recognize the need for more general tools to aid in the design and test of help systems. Therefore we recommend that the tasks under this area include the review of commercial software now available for creating help systems. If these systems seem too limited we would further recommend the expansion of our specific system into a more general set of system design tools. These tools will provide an environment in which other programmers can more easily implement and test help systems.

3. **A series of technical reports evaluating various design strategies for online help menus, screens, and text.** The main goal of this work is to construct a model of the user of online help systems; to design an intelligent help system, we must understand how the user interacts with help systems. To gain this understanding, we will be analyzing the information processing and problem solving behaviors of different users (novice and expert) given two different menu designs and four different types of help text. The data we collect will provide input for the design of our prototype help system as well as help us generate guidelines for the design of help systems in general.

The remainder of this report discusses each of the goals in more detail and summarizes the work accomplished toward each goal.

1 Develop a Handbook of Online Help Guidelines

The goal of this work is to develop a handbook of guidelines and procedures for the design of online help systems. In general, we believe that many handbooks with guidelines do not provide usable information. Too often they give a description of what the final product should look like (e.g., 50% white space, no more than eight commands in a menu, use short, active sentences, etc.), but seldom provide guidance on how to achieve those goals in a complex development environment.

To illustrate, we need simply point to the long history of handbooks and military specifications used for writing manuals. The writing handbooks have been used for centuries, and yet there are still major disagreements between the guidelines found in the handbooks--we continue to find widespread violations of even the most basic guidelines. Similarly, the military has over 450 specifications for the design of technical manuals (some extremely detailed), and yet the usability of these manuals continues to be a major issue.

Most handbooks fail because they give only the guideline and fail to give the following information:

1. Strategies for employing the complex of guidelines are seldom provided. The product--the description of the end result--is given, but there is no guidance on how to get to that end result. It is essential that guidelines be accompanied by procedural information on how to accomplish the goal. Therefore, a product orientation must address the process for achieving that product.

2. A rationale or objective for the guideline is seldom presented, and thus the user has difficulty interpreting just how the guideline should be used. For example, why should there be 50% white space? Can that white space be anywhere or is there something we must understand about how white space affects the information? Similarly, do we use short sentences at all costs or are there some circumstances where short sentences might make comprehension more difficult? The guidelines must be interpreted in the more basic context of how people process information.

3. Guidelines are often presented without limitations; for example, when the guideline is applied, the exceptions to the rule are unclear. It is almost never the case that a guideline can be applied universally, regardless of the other variables involved. Thus it is essential that the limitations in the range of application be presented. These limitations may be in terms of the expertise of the users, the complexity of the application program, or the complexity of the help system. For example, guidelines used for the design of a menu will differ based on whether the menu is used for searching a large database or is used for getting help on different commands in application software.

4. The relative importance of a guideline is seldom indicated. Clearly not all guidelines are equally important, and not all guidelines must be followed with equal precision. Designers are operating in a complex environment and must be able to weigh the importance of various design strategies. Therefore, it is essential that they have a sense of the consequences when they evaluate these alternatives.

5. Examples of how the guideline should be applied are seldom given. This is unfortunate since we know that people learn by modeling; for example, writers implement specifications by looking at a similar book written to the same specification. It is very difficult to take an abstract statement and interpret it in a concrete situation. The goal of guidelines is to teach concepts, and this can be done using the extensive body of literature on the strategies for using examples.

We will focus on both the content and the **design** of the handbook. All of the above factors are critical to the usability of our guidelines. However, addressing all of these issues could easily result in a multi-volume set that few developers would use. The need for presenting information must be weighed against the usability of the document. Our goal is to develop a handbook of guidelines that developers can use on the job. Thus, the document must provide ready access to relevant and usable job information. The tradeoff between the usability of the document and the usability of each particular guideline will be the central focus of our work.

We will use the results from the following task areas to develop the handbook of guidelines:

1.1 Literature Review

Goals

The goal of the literature review is to identify relevant theories, procedures, and guidelines for designing online help. Our work will result in:

1. An online database

2. A report summarizing relevant issues

3. A list of guidelines and procedures for inclusion in the guidebook.

Progress

We began this task by compiling a list of relevant articles to review. To establish the list, we searched journals from the past five years for relevant articles, and we conducted keyword searches of several databases.

Next, we surveyed database programs to find one that would accommodate the literature review. We applied three criteria when we searched for the program:

1. **It had to run a an IBM PC or a Macintosh.** This insured that reviewers contributing to the database (as well as eventual users of the database) would have ready access to the information.

2. **It had to be easy to use.** There are some very powerful database programs available, but most are so complex that we would have to spend significant effort developing a users manual for our application. (DBASE III is an example of a program with this limitation.)

3. **It could not limit field size.** Since we wanted the review to include abstracts of articles, we did not want to restrict the number of words we could use in each field.

The field size criterion turned out to be a major issue in our review of software because almost all database programs place severe restrictions on the size of fields. NoteBook, published by PRO/TEM and selling for $85.00 was the only program that met our criteria; it allows unlimited

field sizes, and designing and modifying the fields requires virtually no expertise. In addition, the program is well-documented and includes a reasonably good help system.

Table 1 lists the fields we are using in our literature review. We have attempted to use a restricted list of keywords and descriptions since the use of unrestricted lists typically proves unwieldy. The allowable keywords and presentation type descriptions are shown in Table 1 under the "keywords" and "type of presentation" fields.

A user of the database can search for articles using any word in any field, but the "keyword" and "type of study" fields are the primary search fields. The keyword field identifies the major feature under consideration, and the type of study field identifies whether the feature is part of an experimental evaluation, a prototype, or is just someone's views. The "independent variable" and "dependent variable" fields are used after the article is accessed. Users can look at these two fields to see if the manipulations are relevant, and thus whether reading the abstract is warranted. The "guideline" fields will feed directly to development of the handbook. The guidelines will be stated to reflect precisely what was reported in the article.

We are currently reviewing the articles at a rate of eight per week, and we have completed 80 reviews so far; the reviews are being compiled in the database program.

Deliverables

The literature review will continue for the duration of the contract, and new articles will be added as they are identified. We will deliver the preliminary database and a paper discussing issues in help design by 1 November 1987.

Table 1: Fields for Literature Review

1. **Bibliographic information**

2. **Keywords**

   user groups
   novice model
   expert model
   user motivation
   user preference

   quick reference
   information categories
   comprehensibility
   multilevel design
   hypertext
   explanation
   functional models/overview

   underlining
   typography
   headings
   graphics
   design guides (manual)
   design guides (on-line)
   windows
   feedback to user

   authoring aids
   command names
   icons
   access mechanisms
   natural language

   screen design
   interface design
   database design

   tutorials
   computer intiated help
   context sensitive
   error handling

   system description
   system evaluation
   cognitive theory
   AI
   evaluation procedures
   hardcopy comparison
   task analysis

3. **Type of presentation**

   experiment
   survey
   speculative

   field observation
   prototype development

   theory
   literature review

4. **Variables--dependent, independent**

5. **Abstract**

6. **Guidelines--proposed, tested**

1.2 Writing Online Information: Expert Strategies

Goals

The goal of this task area is to capture expert knowledge on the design of online help systems. To do this, we have conducted in-depth interviews with writers of online help systems for commercial prodûcts.

Progress

We conducted the interviews to learn how technical writers differentiate between writing and designing online information and writing and designing hardly information. Because there is considerable research and guidance on how to write and design computer manuals, we wanted to know how writers transfer and use those guidelines for designing online help information. For example, we wanted to know:

- How is the design of online information "new"?
- What new procedures, guidelines, and strategies are required?
- What new skills must be learned?
- What new responsibilities writers have?

Ten writers participated in this study, from the following companies and organizations:

- IBM Corporation
- Apple Incorporated
- Microsoft Corporation
- Digital Equipment Corporation
- Symbolics Incorporated
- Information Technology Center at Carnegie Mellon

- Communications Design Center at Carnegie Mellon

The majority of these interviews were completed over the phone, and each session was tape-recorded. The interview sessions followed a specific line of questioning as much as possible; however, each interview progressed differently, based on the responses of the participant. Thus, the questions did not always proceed in the same order.

The survey began with two questions intended to establish prior experience and current duties. Following was a scenario designed to answer our questions about writing online information. The scenario helped place writers in a realistic situation:

```
Assume the following situation:  You need to train a group of
writers to write online information.  These writers are familiar
with computers (through their writing), but they have never
written online information.  The writers have past experience
writing user manuals for [insert product name].  A manual already
exists.  What would you teach them first, second, etc.?
```

Based on responses to the scenario, we progressed through a series of more specific questions such as "is it ever effective to put a hardly manual online?" and "how does screen size affect your writing?" We also focused on:

- Audience concerns.

- Hardware and software issues.

- New guidelines and principles.

We also asked writers about the constraints that affect the design of online information. In particular, we asked how the audience, the hardware, and the application software constrained their design strategies. For example, we asked writers how they would alter their design process when the size of the screen changed. Although most writers viewed screen size as a major constraint on their design process, some still were not sure how they would redesign the information given a different screen size. They agreed the screen design should change, but were uncertain of how. Most also agreed that a variable screen size (e.g., a window under user control) should not be used.

Recommendations

We recommend that these interviews be continued and expanded to include questions about the process by which the help system is developed. We are specifically interested in learning about

the software engineering process in order to identify strategies and procedures for developing online help systems.

Deliverables

We have transcribed and analyzed all of the interviews. A draft report of our findings will be delivered by 1 June 1987. Following the draft report, we will conduct further interviews with developers.

1.3 Help System Benchmarks

Goals

There are two goals in this task area. First, we want to develop a means of describing help systems. At a minimum, this will allow us to identify and compare features of help systems. Ideally, however, it will contribute to the second goal, developing strategies for benchmark testing help systems. Achieving this second goal would give us the capability to objectively compare the effectiveness of alternative help systems.

If we are to develop guidelines for the development and design of online help, we must be able to compare and contrast alternative help systems. However, to the best of our knowledge, there simply is no language or methodology for making such a contrast. While there is considerable research on particular features of help systems, we need a more comprehensive process that will allow us to distinguish between any two help systems. What are the major design parameters for help systems? What is the range of variation across existing help systems on these parameters? There is simply no data on these issues and no methodology for addressing the issue.

While it is essential that we develop a methodology for describing help systems, our ultimate goal is to be able to test the efficacy of alternative systems. We can test features of help systems in the laboratory and identify the relative importance of particular design parameters. However, it is clear that decomposing help "systems" into individual parameters will provide only part of the picture. The help system is a "system" and that system must be subject to comparative testing.

Benchmark testing has a long history, but the importance of the bench marking methodology has been demonstrated most recently in the evaluation of word processing systems. An analysis of word processing was conducted to determine the basic tasks that people using word processors must do regardless of the system they are working on. The tasks were defined in terms of the nature of the edit and the unit of text involved, e.g., delete a word, delete a paragraph, insert a sentence, etc. These core tasks were defined as the benchmark tasks. Any two word processors can be compared in terms of the time it requires to complete each one of these tasks. The test methodology has also been well defined so that when a new word processor is introduced into the market place the procedures and tasks can be used to compare performance on that word processor to previously collected data on other word processors.

Just such a test methodology is the long range goal of this task area. However, identifying legitimate bench marking tasks is a particularly complex issue because the help system is attached to and interacts with particular application software and the goal in using the help system is to accomplish some task with the application software. The problem of comparing help systems for two application software packages in completely different domains, e.g., word processing and data base management, is obvious. However, there is just as much difficulty in dealing with programs within a domain that vary in complexity. For example, the fewer the commands in the application program the fewer the commands required in the help system. Thus even if two help systems had identical designs the one attached to the more basic application software would be easier to use. At least that is how it seems to us at the moment. Grappling with these conceptual issues and developing pilot test strategies will be an ongoing part of the work in this task area.

Progress

We are currently reviewing the help systems for 30 different software packages. The application programs are for operating systems, word processing, data base management, spread sheets, and graphics. A full list of the packages we are reviewing and the system they run on is presented in Table 2. We have now completed the review of 25 of these systems.

The review is serving three immediate goals. First we are building expertise in strategies for designing online help. The variations we are seeing are making us more aware of the strengths and weaknesses of alternative design strategies and are pointing to new design strategies. Second, we are identifying features that are, or are becoming, industry standards--approaches most everyone seems to be taking. Finally, we are developing a scheme for describing help systems. We began with a rough rating/description form and we have modified it as we have gained experience. Our current, and we think final, version is presented in Appendix A.

Deliverables

We will submit a draft report of this work by 1 July 1987. The report will include our final descriptive system (form), each of the 30 help systems described in terms of that form, industry standards in design, and a discussion of more general design issues and design strategies we uncovered as a function of the review. The review will also include our initial thoughts on bench marking, but the major work in that area will begin after this first report is completed.

Table 2
Help systems reviewed

Company	Program
MicroRim	Rbase 4000
Microsoft	Word 3.1 for PC
Microsoft	Word 3.0 for Macintosh
Microsoft	Word 1.05 for Macintosh
VMS/VAX	Operating System
CMU	Emacs
Microsoft	Multiplan 1.10 for PC
Luguru software	Epsilon 3.01
Microsoft	Chart 1.0 for Macintosh
ITC	Andrew Operating System
Satelite Software International	Word Perfect 4.0
Sun Micosystems	Unix Man
Microsoft	Excel 1.0
Lotus Development Corp	Lotus 1-2-3 1A
Protem	Notebook II 2.21
Living Videotext	Thinktank for PC
Ashton-Tate	Framework
Ashton-Tate	Dbase III plus 1.0
DEC	TOPS 20 Operating System
Micropro International	Wordstar 2000+ 1.01
Micropro International	Wordstar 3.3
Harcourt Brace Jovanovich	HBJ Writer 1.1A
Aldus	PageMaker 1.2 for Macintosh
Microsoft	Multiplan 1.01 for Macintosh
SORCIM/IUT Micro Software	SuperWriter
Five systems to be announced	

2 Develop a Toolkit for Designing Help Systems

Goals

The second task area comprises our effort to build help systems in order to test various design strategies. Our goals are (1) to choose a development environment, (2) to choose an application program to design help for, and (3) to implement several help systems for testing.

Progress

Status. We have achieved goals one and two. We are using three Sun 3/50 Advanced Workstations from Sun Microsystems for all our development efforts. The help systems themselves are being implemented in Franc Extended Common Lisp. We have also chosen EMACS our application program. EMACS is a powerful, extensible word processor that is found in most Lisp programming environments (for example, on the lisp machine from Symbolics). Finally, goal three is progressing as demanded by the other task areas. Currently, we have implemented two versions of command help as well as data collection programs that record time-stamped keystroke data for both EMACS and the help system itself.

Considerations. Several factors influenced our choice of a development environment. First, the Sun 3/50 is a powerful workstation with a 19 inch bit-mapped screen and 70 megabyte SCSI hard disk. It runs the Unix operating system and comes with its own window management software. The following points underlie why we chose the Sun:

1. Its widespread use makes it a familiar delivery system, especially in software engineering environments.

2. Its price has fallen in the last year as a result of increased sales. Of the workstations available when we started, it was a best buy.

3. Its window management software runs on stand-alone systems. Our programming efforts, therefore, are not tied to a particular computing environment.

4. Its features allow us to compare more easily help systems that differ along various dimensions (for example, window size, tiled vs. overlapping windows, and mouse vs. keystroke selection).

Our decision to implement help systems using Franc Extended Common Lisp has a more complicated history. Because one goal of the project is to evaluate various design strategies, we originally decided to use a rapid prototyping system. Rapid prototyping systems allows programmers to build and test interfaces very quickly by separating the human-to-computer dialogue from the application program (which actually performs some task). Prototypers allow one to design an interface well before its corresponding application is ready.

After reviewing a number of such systems, we decided that RAPID/USE, a rapid prototyping system from Interactive Development Environments (IDE), seemed to suit our needs. RAPID/USE lets developers directly specify and manipulate a dialogue directly using transition networks. The transition networks are displayed graphically; another program translates these graphs into a intermediate programming language which can then be compiled into executable programs. RAPID/USE also provides links to the IDE's TROLL relational database system.

We purchased the system from IDE and worked with it for approximately three months before deciding it did not meet the project's needs. In part, the problem rested with inadequate documentation. Many apparently simple tasks required many hours to complete merely because the documentation was incomplete, cryptic, poorly organized, and tersely written. But we also had problems with the software itself. For example, database fields could have a maximum of 504 characters, a restriction that posed significant problems for storing help text. We also had difficulty linking the database to the prototyper in the manner shown in the manual. Although IDE was very cooperative, their technical representatives could only help us if we sent them tapes of our programs. Finally, since RAPID/USE is not a full programming language, we would had to have hired a C programmer to add extensions for implementing online help--precisely the situation we intended to avoid by using a rapid prototyper. We sent IDE a letter describing our problems and asking for a refund (a copy of the letter is in appendix 2). IDE suggested alternative solutions, but did refund our money.

Next, we consulted with Allen Newell (Computer Science, Carnegie Mellon) and Robert Williges (Virginia Tech) about prototyping systems. Newell suggested that issues in designing prototyping systems are not well understood. Williges said that almost all prototypers lack generality beyond the domain for which they were designed.

To guarantee that our development environment would provide the features we require, we chose Common Lisp. Common Lisp is fast becoming the standard dialect of Lisp. Its powerful,

interpreted environment yields a fast development cycle. Franc Lisp's Extended Common Lisp is particularly well suited to the smaller Sun 3/50 machine.

Recommendations

We would like to expand our original goals to provide (or recommend) more general toolkit for designing online help. These tools will provide an environment in which other programmers can more easily implement and test help systems. Existing tools may already provide the requisite generality and flexibility. Thus the first phase is to review existing software. From this survey, we will either recommend a software package(s) for designing online help or expand our own help system to provide a more general toolkit.

The first goal is to review the existing software that can be used to design help systems (at least software that exists for various microcomputers). The results of the software review will allow us to assess the current state-of-the-art in development systems. The reviews will also make the guidebook more relevant to its audience--help designers who must deal with the pragmatic aspects of building systems. Even a product review listing should prove beneficial. (The help guidelines could in turn suggests ways to increase the performance of help authoring tools. The interaction between the guidebook and the toolkit will, in effect, lay the foundations for an intelligent authoring system.) We will provide quarterly progress reports of the reviews and analyses of their implications for designing online help.

A number of programs for constructing help systems now exist for the IBM PC and the Apple Macintosh. A representative list of such programs would include the following:

SoftScreen Help...Dialectic Systems, Inc.

Turbo Prolog Toolkit...Borland International

Prototyper...SmethersBarnes

Dan Bricklan's Demo Program...Software Garden, Inc.

MacApp

Course of Action

Guide

The second goal is to begin to plan the structure of the toolkit. Figure 1 shows a possible structure based on our experience with the project's help systems. The overall package is a *help system manager* that comprises several software tools. The manager itself provides links to the application language. In our toolkit, this language will be Common Lisp (designing a stand-alone manager that interfaces with many languages, though possible, presents a more complex software engineering problem). The manager is composed of several tools. The *window (or screen) tool* provides functions for designing screens. The text input component will give designers ways to place text on the screen, whether by typing it, importing it from a file or, ideally, linking it to a help database. A cross-referencer allows screens to be linked other screens, that is, permits the A graphics component might allow the designer to draw lines and boxes or to place bit-mapped graphics on the screen along with the text (this component would be more difficult to realize). Finally, an attributes component allows the designer to specify text attributes such as highlighting, centering, margins, and word wrap (filling specified areas with text) or window attributes such as window size, location, selection device (mouse, key, etc.), and advanced objects (scroll bars, buttons, switches, etc.).

The executive tool provides automatic data collection (time-stamped keystroke data) for the help system, a way to link each help window into one system, an (optional) screen compiler and data compressor (for speed and space efficiency), and a preview module that actually runs the help system in interpreted mode. A further extension to the executive tool might be a compiler that generates stand-alone, executable systems.

Finally, the help database tool provides a way to create, modify, input, and access various structures of help information. For example, one might define certain fields of information for command based help, different fields for task help, and so on. Ideally, the database would link to master windows so that instances of the master windows would inherit a field structure and formatting.

The above description of the general toolkit is only an example of what we might do. If necessary, we will develop a more detailed proposal and cost estimate. The software review, however, is within the scope of the current project.

Deliverables

The main deliverable from this task area is the specific help systems we have designed for EMACS. If we adopt the expanded goals, the deliverables would include a review and analysis

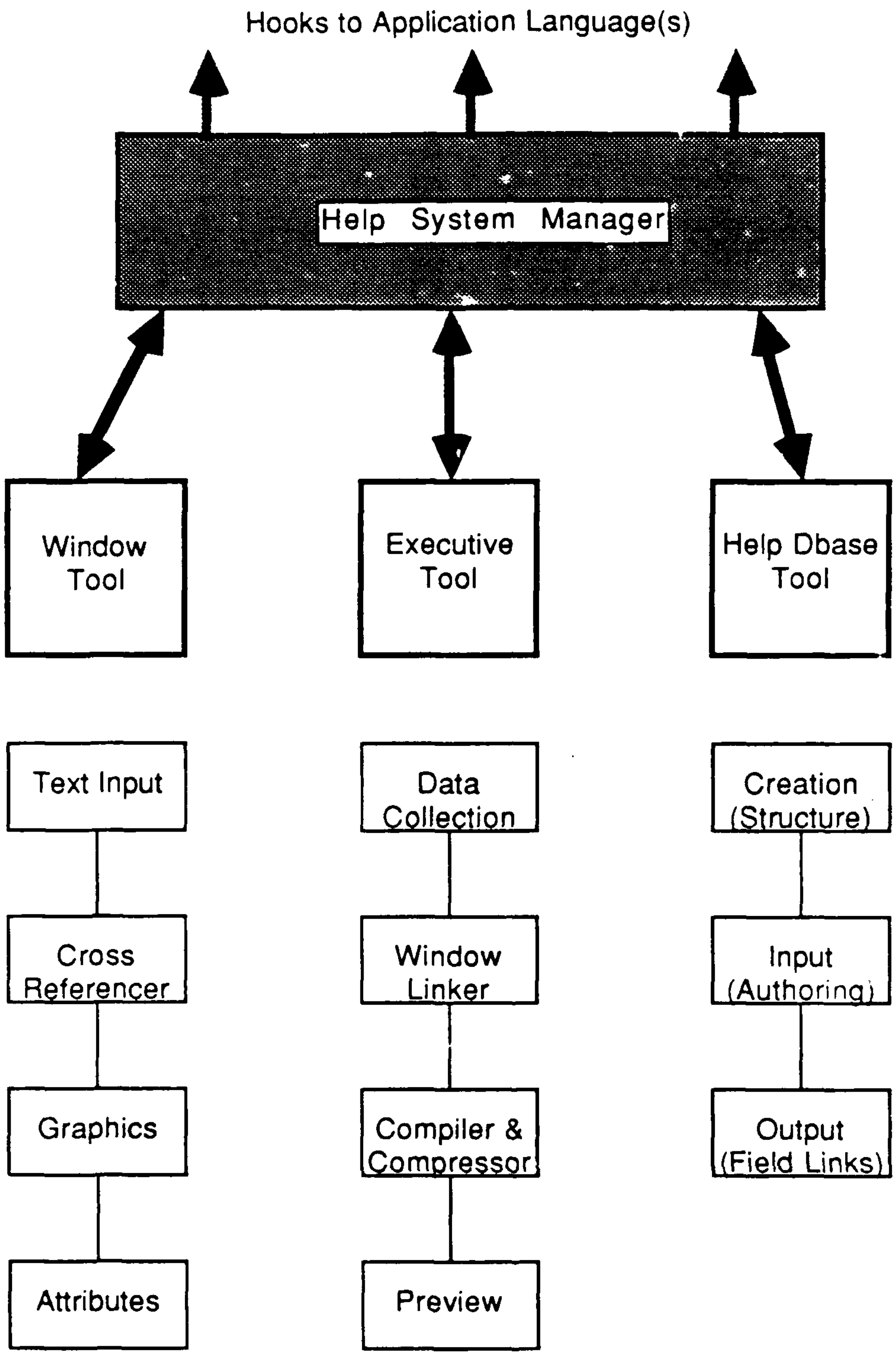

Figure 1. Structure of a Help System Manager

of the current state-of-the-art in help design software and, optionally, a generalized toolkit for help design based on our own implementation strategies. Work on these deliverables will continue throughout the duration of the project.

3 Evaluate Design Strategies

The third area of our work involves the empirical evaluation of alternative strategies for designing online help menus, screen designs, and help text designs. The primary goal of this work is to build a model of the user of online help systems. If we are to provide intelligent help to the user we must understand how the user approaches and interacts with the help system. Users go to a help system because they lack the necessary knowledge about how to accomplish some task with the application software--the individual must learn. Thus a help system is really a computer based instructional system. It must "teach" the person what to do. When we say "teach" most people will think of a tutorial or some full blown instructional package. In some circumstances a tutorial may be necessary or at least useful. However, in most cases other approaches to teaching will be more appropriate. For example, the person may simply require master-apprentice guidance with the task or simply a reference available while the task is attempted.

In any instructional design effort it is essential that the instruction begin with what the person knows, that is, a representation of the problem in familiar terms, and progress to the needed information. If we are to provide intelligent instruction then we must have a more comprehensive model of the learner. That is, we must understand the way in which they represent the knowledge requirement and how they select strategies for obtaining the necessary knowledge. We must understand how they process the information presented and we must understand the "bugs" in their thinking about the problem.

The design of the entire help system is driven by the user model. Ideally a complete model of the user would allow for an intelligent help system that would volunteer help as it deemed necessary. However, a model of the user is essential for all aspects of design. For example, designing the help menu must link to the way the person describes or represents the problem to him or her self. A person may think in terms of the ultimate real world goal quite independently of the software: "I have to exchange these two paragraphs." Another person may think in terms of computing tasks ("I need to cut and paste") while still another may focus directly on the command he or she thinks is relevant to the task ("I think I need ctrl b").

The above alternative representations vary simply in terms of the label the person gives to the task. However, it may well be that different users think of a given goal as involving entirely different sets of tasks. It would be very unusual, for example, for the computing tasks required to accomplish a goal to be identical to the tasks one would do without using the computer. This is certainly true for editing, database management, and virtually any system control task. In essence the computer representation requires a reconceptualization of the tasks involved.

An effective help system must meet the user in terms of his or her representation of the task and provide a path to the appropriate information. Of course determining what is "appropriate" is another major issue. Who should determine--the computer or the user--when tutorial, quick reference, or some intermediate level of contextualized help should be presented. Regardless of who determines it, what factors are related to the effectiveness of each type of help.

The goal of this work is to analyze how users represent problems, search for help, and use help information. We will be analyzing the information processing and problem solving behaviors of the users given different screen designs and given different help information. We are particularly interested in the differences in processing and problem solving as a function of the users background and the type of task.

This work is being done in our prototyping environment i.e, using EMACS on the Sun work stations. We are currently conducting two evaluations and are planning a third.

3.1 Problem Representation, Learning, and Menu Design

We have several goals in conducting this evaluation. When users encounter a problem using application software, they develop a representation of that problem, refine the representation in terms of the particular resources available, and then select a strategy for getting the information necessary to solve the problem. The goal of this evaluation is to understand how problems are represented and how strategies are selected. This data will aid us in the construction of a model of a problem solving process that we hope will have broad impact on the interface design process as well as provide specific input for the design of our own help system.

We are looking at the problem solving processes of users at different levels of experience and of tasks at differing levels of complexity. There are two groups of users: those with an intermediate level of experience with EMACS and those with no prior EMACS experience (but with word processing experience). In a session they will be given tasks at three levels of

complexity: easy (delete a letter), intermediate (search and replace), and complex (move a paragraph from one file to another).

Perhaps most importantly, we are studying how the menu design affects the problem solving process and how that process changes with practice. We expect that the menu design will have a significant impact on the problem solving strategies of novice users and that practice will exaggerate rather than level the differences. That is, the structure of the menu should play a significant role in the development of the users functional model of the application software, that is, how the user thinks the application software works. That model should be learned simply through repeated use of the menu and eventually begin to guide how the user represents problems and thinks about information needs. To the extent that the menu provides a functional representation of the system, it will facilitate the learning of the system. To the extent that the model is not functional, it will not facilitate and may even interfere with learning.

We will be comparing two menus: an alphabetical menu and menu in which the commands are grouped into categories related to editing tasks and the categories are labeled. The alphabetical menu and the task-oriented menu are found in Appendix B. While the alphabetic menu may look unwieldy, it is in fact the most common type of help menu we have found in our review of existing systems. We have found alphabetic menus containing well over 100 commands. In this study we are restricting the command set to 58.

In summary, the study will involve four independent variables: menu type (alphabetic or functional groups); user experience (word processors with or without EMACS experience); sessions (we will look at learning across four sessions); and editing task (there will be three editing tasks at each of three levels of difficulty in each session).

In each session the users are given a hard copy text with an editing change marked. They are asked to describe the editing task that is required. Next they are asked to turn to the help menu and identify the commands they think they will use to accomplish the editing task. Finally they are given 10 minutes to complete the edit. During the editing they have access to the help menu and to a command based help system. There are nine such trials during each session.

The time and accuracy in completing the edit will be analyzed to determine the effects of menu design on learning the editing tasks. The performance of the experienced users will serve as a benchmark against which novice learning can be compared. In terms of task representation we are particularly interested in whether novice representations begin to approximate the

experienced users and whether the task-oriented menu design aids in learning the commands, in learning to use EMACS most effectively, and in representing the tasks in the same way as the expert.

We will collect all the evaluation data by 15 July 1987 and deliver a draft report of the findings by 15 October 1987.

3.2 Procedures for Defining a Menu Structure

In this task area we are examining alternative methods for defining the menu structure and evaluating the effectiveness of the alternative structures. This work will contribute both procedures and design guidelines for help menus.

A menu for a help system is a mechanism for accessing information and, as discussed above, it may also contribute to the development of the users' conceptual model. As an effective access mechanism, the menu should reflect the way the user thinks about the system. Clearly, a random ordering of commands would be extremely difficult to use and an alphabetical menu would be effective only if the individual knew the command for which help was required. How should a menu be designed to capture this representation? What is the methodology a developer should employ to identify how users' represent tasks? We can perhaps capture that representation by asking the users to describe a variety of tasks. That is, the task representation methodology described in the last task area may well provide us with the data necessary to organize and label the help menu in such a way as to capture the users representation. However, our concern is with developing a methodology that not only can be used but will be used in the software engineering environment. We suspect that the task representation methodology would not be widely used simply because it is complex and time consuming. We will therefore consider the tradeoff of simply using expert document design judgment as a basis for the menu design.

Thus far we have focused on the methodology for designing a menu to capture the user's task representation. Next we must consider the methodology for developing a menu that reflects the structure of the application software, that is, a useful model for thinking about the set of commands and their uses. We are now concerned with representing the structure of the application software, not necessarily the structure of the real world tasks. Hence the methodology must focus on software rather than the real world tasks.

Of course, the structure of the application software might best be captured simply through the organization of the command set in the software documentation. How did the system designer see the structure of the system? Thus we will construct a menu based on the organization of the commands in the documentation for Gossling's EMACS. However, while this may represent the structure of EMACS it may not be a functional structure for the user of EMACS. That is, it may represent the program structure rather than the conceptual structure of the system.

Thus we will employ yet another methodology to attempt to capture the functional representation of EMACS. All 110 EMACS commands (command names and keystrokes) are typed on separate index cards. These cards are then given to individuals with instructions to sort the cards into groups that "make sense" and in such a way that you could put the same cards back together again if requested to. Once the categories are formed (usually users create between 4 and 10 categories), the user is asked to describe or label the group.

The organization of the cards in the categories is then taken to reflect the users functional organization of the command set for the application software. For example, this card sort methodology has been used to look at the functional model airplane mechanics have of the airplane. Photographs of parts of the plane were put on cards and mechanics were asked to sort them. It was found that relatively new mechanics sorted the parts into categories based on physical proximity while experienced mechanics placed the parts into groups based on function.

In summary we will be comparing four methodologies for creating help menus: task representation, card sort, computer scientist judgment, and document designer judgment. The task representation and card sort strategies, however, require a specification of the users who will be tested. Thus for each of these two methodologies we will look at menus generated based on testing of experts and based on testing of novices (EMACS novice but word processing knowledgeable).

We have already collect the card sort data from 10 expert, intermediate, and novice users and are analyzing the data. We are comparing cluster analysis, non-metric scaling, and linear discriminant analysis methodologies for analyzing the data. Initial indications, however, are that we may need to increase the number of users in each group to get more stable data regardless of the methodology.

We have just begun to collect the task representation data as described in the previous task area. Both this and the card sort data will be collected and the analysis completed by 1 August 1987. A draft report of the findings will be submitted by 1 October 1987.

The first part of the work in this task area is to develop the methodology for creating menus. Once we have completed the menu designs, we will compare the effectiveness of the alternate designs. We are currently developing the methodology for comparing the effectiveness of the alternative menus. The design will be completed and testing begun on 1 October 1987.

3.3 Defining the Content and Design of Help Information

The last task area focuses on the help information to be presented to the user. The goal is to understand the characteristics of the task, the context, and the user that determine the nature of the help information required. The work will address the following kinds of questions. Under what conditions should help be brief and telegraphic and under what conditions should it be elaborated upon? Is a fully cohesive text, i.e., regular discursive writing necessary to communicate the complexity of tasks or can a novice user get the necessary information from terse, highly formatted help text? What is the role of examples; how particularized must the examples be to the user's *current* work state? Answering questions such as these is essential to the design of context sensitive help.

The initial work in this task area has been on defining and developing different help data bases. Initially we are concerned issues of content and format specification. Later we will turn to strategies for using the computing power to, for example, study hypertext or embedded menu strategies and interactive help.

During the first year we have defined three types of help databases. We will complete the databases by 1 August 1987. Once developed, these different help texts will be used to address the issues and questions described above.

The most basic help text is command oriented help. Separate help is provided for each EMACS command: each command on the menu is associated with its own help text. The help defines the command, describes its function, presents any warnings concerning common problems, and identifies related commands. This is the standard kind of help one finds with most application software. Appendix C presents an example of this help (Sample Command Help).

The command help will be contrasted to the help found in the documentation for EMACS. This is command-based help written by the computer scientist who developed the software. Appendix C gives an example of this help (Sample from Online Manual).

Presenting command based help fails to contextualize the help information. There is no contextualization in terms of the editing task nor in terms of the set of commands. This of course presents two types of problems. First, the user may understand the description of the command but still not be certain whether or not it applies in the particular situation. Second, even if it is decided that it does apply, there is uncertainty of exactly how it is used in relation to other commands. We are therefore building two other types of help texts that attempt to provide the necessary context. The two strategies differ radically in design philosophy.

Task-oriented help presents a complex of related commands. The related commands might be part of a procedure, e.g., the sequence required for cut and pasting, or may represent different levels of strategy for accomplishing a task e.g., different strategies for searching and replacing. The commands are defined and the functions described. Most importantly a range of examples are available illustrating when and how one might want to use the commands. The information is all presented in a very terse, highly formatted structure. The philosophy is that while we want to help contextualize information, the user wants to accomplishing the editing task--not read a lot of help text. Appendix C presents the task-oriented help (Sample Task Help).

The final strategy for presenting help is to present discursive help. This approach reflects the design strategy found in help for systems that have large screen displays, e.g., help for the Symbolics interface and for the Andrew interface developed at Carnegie Mellon. It is also consistent with the philosophy links hard copy and online documentation/help.

The discursive help text was written by a professional writer experienced in developing Andrew help text. The Information Technology Center (the architects of the Andrew system) describe the help that was written as a prototype of the Andrew help for EMACS. It is a continuous text that the user scrolls through. Thus there is not a "page format" as found in the task-oriented help. Also, the text because of its discursive format, provides the full range of cohesive devices of a well structured text to enable the writer to build an understanding. The discursive text explains in detail, whereas in the task-oriented help the user must infer relations and fill in some basic information. Appendix C shows an example of discursive help (Sample Discursive Help).

Appendix A

Company:
Program:
Hardware used for review:
Version:
Date:
Type:
Size of program (in k bytes):
Size of help system (in k bytes):

General description

The help system is

- ☐ An online document
- ☐ A segmented database

Are there instructions on how to use the help system?

- ☐ Yes
- ☐ No

Does the help system provide feedback/ error messages on help related actions?

- ☐ Yes (describe)
- ☐ No

Access

Access to the help system is gained

- ☐ Through a command, describe:

- ☐ Through a menu selection, how are items selected?

Is the help access context sensitive?

- ☐ Yes
- ☐ No

If yes, describe

Finding a topic

Once in the help system, what topic search facilities are provided?

- ☐ Page up/down
- ☐ Scrolling (between topics only)
- ☐ Go to line x
- ☐ Go up/down x lines
- ☐ Find string (text search)
- ☐ Find keyword
  Describe the convention used to name keywords
- ☐ Hypertext
  Describe the hypertext system
- ☐ Menu
- ☐ Trace steps
- ☐ Return to Point of entry/Top menu
- ☐ Other:

Menu design

If the help system is menu based, answer these:

How many levels of menus are there?

How many items are there on the top level menu?

What is the average number of items per menu at lower levels?

What is the total number of menu screens?

How are the menu items named?

- ☐ command based names
- ☐ task based names
- ☐ other (describe)

What is the basis for the hierarchy of menus?

How are the items within a menu organized?

How are menu items selected?

Topics

How many help topics are there?

What is the average number of words per topic?

What is the average number of screens per topic?

What kinds of help are provided

- ☐ syntax
- ☐ examples
- ☐ bugs
- ☐ troubleshooting
- ☐ possible applications
- ☐ options (shortcuts)
- ☐ functions
- ☐ warnings
- ☐ definition of topic in task domain
- ☐ other (describe)

Are the kinds of help separated into discrete chunks?

Are the kinds of help provided consistently across topics?

Subtopics

Does the system provide access to subtopics?

- ☐ Yes
- ☐ No

If yes, how are the subtopics accessed?

- ☐ Menu (describe organization and naming conventions for items)
- ☐ Page up/down
- ☐ Scrolling
- ☐ Go to line x
- ☐ Go up/down x lines
- ☐ Find string (text search)
- ☐ Find keyword
  Describe the convention used to name keywords
- ☐ Hypertext
  Describe the hypertext system

Formatting

The formatting is
- ☐ window based
- ☐ text based

Screen size:

Default size of help window:

Average lines of text per screen:

Average number of words per screen:

The help text
- ☐ Replaces the working window
- ☐ covers a portion of the working window (describe, how much, what part is covered)

Does the system allow manipulation of the help window?
- ☐ Yes
- ☐ No

If yes, describe (resizing? repositioning? control of fonts or character sizes?)

Does the text for each topic fit on one screen?
- ☐ Yes
- ☐ No

If no, describe mechanisms for viewing all of text
- ☐ Page up/down
- ☐ Scrolling (between topics only)
- ☐ Go to line x
- ☐ Go up/down x lines
- ☐ Find string (text search)
- ☐ Find keyword
  Describe the convention used to name keywords
- ☐ Hypertext
  Describe the hypertext system
- ☐ Menu

Can you work on the program while viewing the help?

- ☐ Yes
- ☐ No

Evaluate the following (1 is below average, 3 is above avaerage)

1 2 3	consistent layout between topics
1 2 3	consistent layout between types of help
1 2 3	visual cues used (highlighting, bullets, indenting, etc.)
1 2 3	cross references called out
1 2 3	graphics in text
1 2 3	clearly visible headings
1 2 3	adequate use of blank space
1 2 3	discrete visual chunks

Language

Comment on these aspects of the language used

- complex sentences
- undefined terms and jargon
- wordiness
- noun strings
- active and passive voice
- personal pronouns
- parallelism
- clarity of writing
- the use of paragraphs and graphics
- meaningful, task based headings

Integration with hardcopy

Comment on the relationship between hardcopy documentation and online help. Are there instructions (hardcopy or online) that tell when to refer to each? Is the information redundant?

Integration with program	Comment on the compatibility of the help system with the surrounding software. Are the controls (methods of issuing commands) similar? Is the complexity of the help system comparible to the complexity of the program?
System Response time	Approximate time to access help system from program: Approximate time to access a topic while in system:

Summary/Evaluation

Appendix B

Emacs Commands

i. Introduction

1. Abort command(s) ^g

2. Back character ^b
3. Back sentence Esc a
4. Back word Esc b
5. Beginning of document Esc <
6. Beginning of line ^a
7. Beginning of window Esc ,

8. Close file
9. Cut region (Wipe) ^w
10. Cut to end of line (Kill) ^k
11. Cut to new file Esc ^w

12. Delete character at cursor ^d
13. Delete character back ^h
14. Delete word back Esc h
15. Delete word forward Esc d

16. End of document Esc >
17. End of line ^e
18. End of window Esc .
19. Enlarge window ^x z
20. Erase region (no paste)
21. Exit Emacs ^c

22. Forward character ^f
23. Forward sentence Esc e
24. Forward word Esc f

25. Insert file ^x ^i

26. Line down (Next) ^n
27. Line up (Previous) ^p
28. List open files ^x ^b

29. Mark beginning of region ^@
30. Move line to top of window Esc !

31. Open file (Visit) ^x ^v

32. Paste (Yank) ^y

33. Redraw display ^L
34. Remove other windows ^x 1
35. Remove this window ^x d
36. Rename and save file ^x ^w
37. Replace with file ^x ^r
38. Run Unix command ^x !

39. Save file ^x ^s
40. Save files and exit ^x ^f
41. Save modified files only ^x ^m
42. Scroll 1 line down ^z
43. Scroll 1 line up Esc z
44. Scroll 1 page down ^v
45. Scroll 1 page up Esc v
46. Search back (Reverse) ^r
47. Search forward ^s
48. Search/replace all Esc r
49. Search/replace some (Query) Esc q
50. Shrink window ^x ^z
51. Split window ^x 2
52. Switch to new file ^x b
53. Switch to open file ^x o

54. Undo command(s) ^x ^u

55. Window down (Next) ^x n
56. Window up (Previous) ^x p

Enter number of topic or i for introduction:_

Emacs Commands

Basic Cursor Movement

1. Forward Character ^f
2. Back Character ^b
3. Line Up (Previous) ^p
4. Line Down (Next) ^n
5. Beginning of Line ^a
6. End of Line ^e
7. Beginning of Window Esc ,
8. End of Window Esc .

Scrolling

9. Scroll 1 page down ^v
10. Scroll 1 page up Esc v
11. Scroll 1 line down ^z
12. Scroll 1 line up Esc z
13. Move line to top of window Esc !

Deleting

14. Delete character at cursor ^d
15. Delete character back ^h
16. Delete word forward Esc d
17. Delete word back Esc h

Working with Files

18. Save file ^x ^s
19. Rename and save file ^x ^w
20. Save files and exit ^x ^f
21. Save modified files only ^x ^m
22. Open a file (Visit) ^x ^v
23. Close file Esc x delete-buffer
24. Insert file ^x ^i
25. Replace with file ^x ^r
26. List open files ^x ^b
27. Switch to open file ^x o
28. Switch to new file ^x b

Miscellaneous Commands

29. Exit Emacs ^c
30. Abort command(s) ^g
31. Redraw display ^L
32. Undo command(s) ^x ^u
33. Run Unix command ^x !

Accelerated Cursor Movement

34. Beginning of Document Esc <
35. End of Document Esc >
36. Forward Word Esc f
37. Back Word Esc b
38. Forward Sentence Esc e
39. Back sentence Esc a
40. Forward Paragraph Esc]
41. Back paragraph Esc [

Searching and Replacing

42. Search forward ^s
43. Search back (Reverse) ^r
44. Search/replace all Esc r
45. Search/replace some (Query) Esc q

Cutting and Pasting

46. Mark beginning of region ^@
47. Cut region (Wipe) ^w
48. Paste (Yank) ^y
49. Cut to end of line (Kill) ^k
50. Cut to new file Esc ^w
51. Erase region (no paste)

Working with Windows

52. Window up (Previous) ^x p
53. Window down (Next) ^x n
54. Split window ^x 2
55. Remove this window ^x d
56. Remove other windows ^x 1
57. Shrink window ^x ^z
58. Enlarge window ^x z

Other Help

i Introduction to Emacs

Type the number of a topic for help or i for introduction:_

Appendix C

Sample Command Help

Command Name:	split-current-window
Key strokes:	Control x 2
Description:	Splits the current window into two windows.
Function:	Invokes the two-window mode. When you split the window, screen space is divided evenly between the two windows. If you have text or a buffer visible on the screen, it will appear in both windows. That is, you can access your entire file, or buffer, through each of the windows. This is useful if you want to look at two areas of the same file. NOTE: You can divide the screen into more than two windows. depending on available screen space. Generally, you'll find that it's only convenient to look at two windows at once.

Press any key to continue:

Sample from Online Manual (Gosling)

3. Buffers and Windows

There are two fundamental objects in EMACS, buffers and windows. A buffer is a chunk of text that can be edited. It is often the body of a file. A window is a region on the screen through which a buffer may be viewed. A window looks at one buffer, but a buffer may be on view in several windows. It is often handy to have two windows looking at the same buffer so that you can be looking at two separate parts of the same file, for example, a set of declarations and a piece of code that uses those declarations. Similarly, it is often handy to have two different buffers on view in two windows.

3.1. Manipulating Buffers

buffer-size
Returns the number of characters in the current buffer.

current-buffer-size
Returns the current buffer name as a string.

delete-buffer *buffername*
Deletes the named buffer. All text that it contains will be thrown away. Deleting a buffer will only fail if it has an attached process that is actively running.

^X^B **list-buffers**
Produces a listing of all existing buffers giving their names, the name of the associated file (if there is one), the number of characters in the buffer and the indication of whether or not the buffer has been modified since it was read or written from the associated file.

pop-to-buffer *buffer-name*
Switches to a buffer whose name is provided and ties that buffer to a popped-up window. Pop-to-buffer is exactly the same as switch-to-buffer except that switch-to-buffer ties the buffer to the current window; pop-to-buffer finds a new buffer to tie it to.

Sample Task Help

Name: Creating windows

Level: Novice

Commands: Control x 2 -- creates 2 windows
Control x 1 -- goes back to 1 window

Function: Windows allow you to split the screen into sections. Each section is called a window. You can use windows to look at more than one file at the same time or to look at different parts of the same file. You can also use windows to work with programs outside of Emacs (without leaving Emacs).

Example:

1. Hit Control x 2 (The screen will split into two windows.)

2. Hit Control x 1 (The screen should return to one window.)

Applications:

General

1. Look at mail while in Emacs

2. Look at help while in Emacs

3. Look at bulletin boards while in Emacs

4. Look at system messages while in Emacs

Sample Discursive Help

Buffers, Files, and Windows

Buffers and Files. Whenever you edit a file in Emacs, Emacs makes a copy of the file for use in your editing session and holds this file copy in a "buffer." A buffer is a workspace in memory which is controlled by the Emacs program. A buffer contains a copy of an original file and the original file remains untouched in permanent storage. When you place a file copy in a buffer, the contents of the buffer (the copy) are said to be "associated" with the original file.

Buffers are very important because they let you modify (edit, experiment with, change) a copy of a file without affecting the original file. After you modify a buffer, you can:

-Change the original file by saving the modifications you have made in the buffer. Because the buffer is associated with the original file, Emacs knows where to send the changes.

-Create a new file by saving the contents of a buffer to a different file name. This allows you to associate the buffer with a different file and save the buffer contents there instead..

-Throw away the buffer by not saving it. Do this when you don't like the modifications you have made and you want to quit editing or to start again with a new copy of the original file.

Windows. You can look at and edit the contents of a buffer through a "window." When you use a window to look at a buffer, the buffer is said to be "associated" with the window. That is, Emacs understands that the actions you take in that window should affect that specific buffer.